TREE STRUCTURES FOR ORTHOGONAL TRANSFORMS AND APPLICATION TO THE HADAMARD TRANSFORM

Martin VETTERLI

Laboratoire d'Informatique Technique, Ecole Polytechnique Fédérale de Lausanne, 16 *Chemin de Bellerive, CH-*1007 *Lausanne, Switzerland*

Received 28 May 1983
Revised 27 June 1983

Abstract. Tree structures for computing orthogonal transforms are introduced. Two cases, delay trees and decimation trees, are investigated. A simple condition, namely the orthogonality of branches with a common root, is shown to be necessary and sufficient for the overall transform to be orthogonal. Main advantages are structural simplicity and a number of operations proportional to $N \text{Log}_2 N$.

Application of the tree structures to the Walsh–Hadamard Transform (in natural, sequency and dyadic order) is presented. A single module can be multiplexed or used in parallel in order to perform all operations. Such a system is shown to be well suited for hardware implementation.

Zusammenfassung. Baumstrukturen zur Berechnung orthogonaler Transformationen werden eingeführt. Zwei Fälle, Verzögerungsbäume und Dezimierungsbäume, werden untersucht. Eine einfache Bedingung, nämlich die Orthogonalität von Aesten mit gemeinsamer Wurzel, wird als notwendig und genügend gefunden damit die resultierende Transformation orthogonal ist. Grösste Vorteile sind strukturelle Einfachheit und eine Anzahl Operationen die zu $N \text{Log}_2 N$ proportional ist.

Anwendung der Baumstrukturen zur Berechnung der Walsh-Hadamard Transformation (in natürlicher, sequentieller und dyadischer Ordnung) wird vorgestellt. Alle Operationen können mit einem enzigen Modul, dass multiplexiert oder parallel geschaltet wird, ausgeführt werden. Es wird gezeigt dass ein solches System für eine 'hardware' Implementierung gut geeignet ist.

Résumé. Des structures en arbres pour l'évaluation de transformations orthogonales sont introduites. Deux cas, des arbres à délais et des arbres à décimation, sont analysés. Une condition simple, l'orthogonalité de branches à racine commune, est démontrée comme nécessaire et suffisante afin d'obtenir une transformée totale orthogonale. Les avantages principaux sont la simplicité structurelle et un nombre d'opérations proportionnel à $N \text{Log}_2 N$.

L'application des structures en arbres au calcul de la transformée de Walsh–Hadamard (en ordre naturel, sequentiel et dyadique) est présentée. Un module unique peut être multiplexé ou utilisé en parallèle afin d'exécuter l'ensemble des calculs. Il est montré qu'un tel système s'adapte bien à une implantation 'hardware'.

Keywords. orthogonal transforms, tree structures, fast Walsh–Hadamard transform, hardware implementation of Walsh–Hadamard transform.

1. Introduction

Generalized orthogonal transforms have been proposed [1] where particular cases are the Fourier and the Walsh–Hadamard transform. In the following, a new class of orthogonal transforms is defined. The topological characteristic is that their flow-graphs are tree structured.

Definition and analysis of the tree structures is done in Section 2. Two particular cases which are interesting for applications, the binary delay tree and the binary decimation tree, are investigated. A simple condition, namely the orthogonality of branches with a common root, is shown to be necessary and sufficient for the overall transform to be orthogonal. Interesting features of these transforms are their inherent simplicity and the fact that they can be computed with an order $N \operatorname{Log}_2 N$ operations (an usual lower bound for fast transforms).

Both the Discrete Fourier Transform (DFT) and the Walsh–Hadarmard Transform (WHT) can be computed using the tree approach. Since the DFT case has been analysed elsewhere [2], we focus in Section 3 on the implementation of the WHT with decimation and delay trees. The computation of the WHT in various orderings (natural, sequency and dyadic) leads to simple modular structures. In particular, the WHT in dyadic order results in a regular structure with identical modules. In all cases, the number of adds and subtracts is equal to $N \operatorname{Log}_2 N$ as for other fast WHT's.

Finally, Section 4 discusses some hardware aspects of the tree model. Emphasis is placed on modularity. It is seen that arbitrary speed can be attained with parallelism, or minimum hardware with multiplexing; all that using a single module with only some interconnection or buffering details being modified.

2. Tree structures for orthogonal transforms

The analysis is restricted to binary trees because on the one hand, they are most useful for applications and on the other hand, generalization to other radices is immediate.

Binary delay trees

Consider a filter tree as shown in Fig. 1 which can either perform a running transform or whose output is only computed every $2N$ samples in the case of a block transform. $x_0, x_1, \ldots, x_{2N-1}$ are the input samples and $y_0, y_1, \ldots, y_{2N-1}$ the corresponding transform values. B_1 and B_2 are two N-dimensional orthogonal transforms where N is usually a power of 2. $\boldsymbol{B}$ is a matrix with B_1 and B_2 blocks on the diagonal and I_N off diagonal.

In the following it will be shown by recursion that a filter tree as in Fig. 1 performs an orthonormal transform.

First we prove that if the matrix $\boldsymbol{A}$ defined by (1) is orthogonal, then the transform of dimension $2N$ is orthogonal.

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}. \tag{1}$$

The term orthogonal matrix refers usually to orthonormal lines resulting therefore in orthonormal columns as well [3]. A weaker condition is sufficient in the case analysed here: the lines of the matrix have to be orthogonal. This is expressed by (2) where Λ_a and Λ_b are arbitrary diagonal matrices.

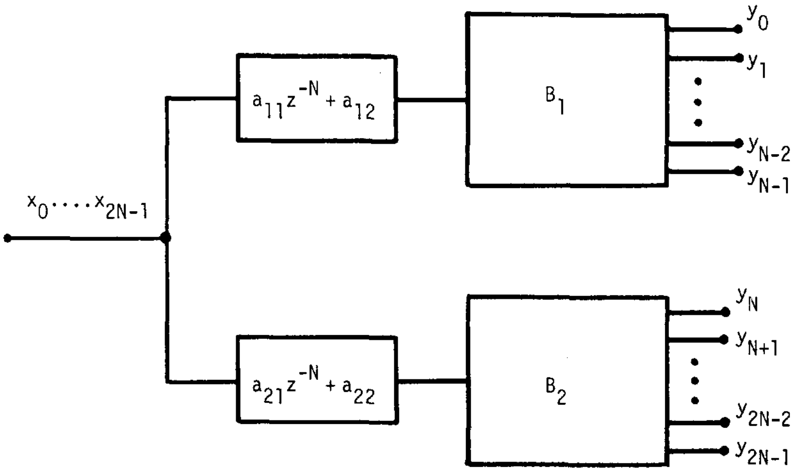

Fig. 1. Delay tree. B_1 and B_2 are N-dimensional orthogonal transforms.

$$AA^T = \Lambda_a$$
$$BB^T = \Lambda_b. \qquad (2)$$

The first set of outputs is given by (3) where I_N and 0_N are identity and zero matrices of dimension $N \times N$.

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \\ y_N \\ \vdots \\ y_{2N-1} \end{bmatrix} = \left[\begin{array}{c:c} B_1 & 0_N \\ \hdashline 0_N & B_2 \end{array}\right] \cdot \left[\begin{array}{c:c} a_{11}I_N & a_{12}I_N \\ \hdashline a_{21}I_N & a_{22}I_N \end{array}\right] \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \\ x_N \\ \vdots \\ x_{2N-1} \end{bmatrix} \qquad (3)$$

Using the Kronecker product, (3) can be rewritten as in (4).

$$y = B \cdot (I_N \times A) \cdot x = C \cdot x. \qquad (4)$$

One can show that the overall transform is still orthogonal or equivalently, that C times its transpose is diagonal. Using properties of the Kronecker product [4] and of block matrix operations, one gets (5).

$$\begin{aligned} C \cdot C &= B \cdot (I \times A)(I \times A)^T \cdot B^T \\ &= B \cdot (I \times AA^T) \cdot B^T \\ &= \begin{bmatrix} B_1 & 0_N \\ 0_N & B_2 \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 I_N & 0_N \\ 0_N & \lambda_2 I_N \end{bmatrix} \cdot \begin{bmatrix} B_1^T & 0_N \\ 0_N & B_2^T \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1 \cdot \Lambda_{B_1} & 0_N \\ 0_N & \lambda_2 \cdot \Lambda_{B_2} \end{bmatrix} = \Lambda_C. \end{aligned} \qquad (5)$$

The initial step of the recursive proof is trivial. A two dimensional transform will be orthogonal if the two branches are orthogonal. Obviously, the orthogonality of the branches is also a necessary condition.

Therefore, a binary delay tree performs an orthogonal transform if and only if each sublock is orthogonal.

Decimation trees

A decimation tree is a structure where each root is followed by N branches which are subsampled by N. One immediate question is: can the original signal be reconstructed from the N subsampled outputs? This general question is answered before turning to the particular case of binary orthogonal trees.

Assume a system as depicted in Fig. 2 where $H_i(z)$ and $F_i(z)$ are length-N FIR filters as in (6).

$$H_i(z) = h_{i,N-1}z^{-(N-1)} + \cdots + h_{i,1}z^{-1} + h_{i,0}. \qquad (6)$$

If we introduce the matrix H whose lines are the coefficients of the filters, we get the output of the decimator in (7).

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} h_{1,N-1} & h_{1,N-2} & \cdots & h_{1,0} \\ h_{2,N-1} & h_{2,N-2} & \cdots & h_{2,0} \\ \vdots & \vdots & & \vdots \\ h_{N-1,N-1} & h_{N-1,N-2} & \cdots & h_{N-1,0} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \qquad (7)$$

The output $\hat{x}$ is computed in (8) with a matrix whose columns are the impulse responses of the FIR interpolation filters.

$$\begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \vdots \\ \hat{x}_{N-1} \end{bmatrix} = \begin{bmatrix} f_{0,0} & f_{1,0} & \cdots & f_{N-1,0} \\ f_{0,1} & f_{1,1} & \cdots & f_{N-1,1} \\ \vdots & \vdots & & \vdots \\ f_{0,N-1} & f_{1,N-1} & \cdots & f_{N-1,N-1} \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} \qquad (8)$$

It is clear that reconstruction is always possible if the matrix H is nonsingular which is equivalent to say that the N impulse responses of the input filters have to span the N dimensional signal space. The interpolation filters follow simply from (9) and $\hat{x} = x$ as shown in (10)

$$F = H^{-1}, \qquad (9)$$
$$\hat{x} = F \cdot y = F \cdot H \cdot x = x. \qquad (10)$$

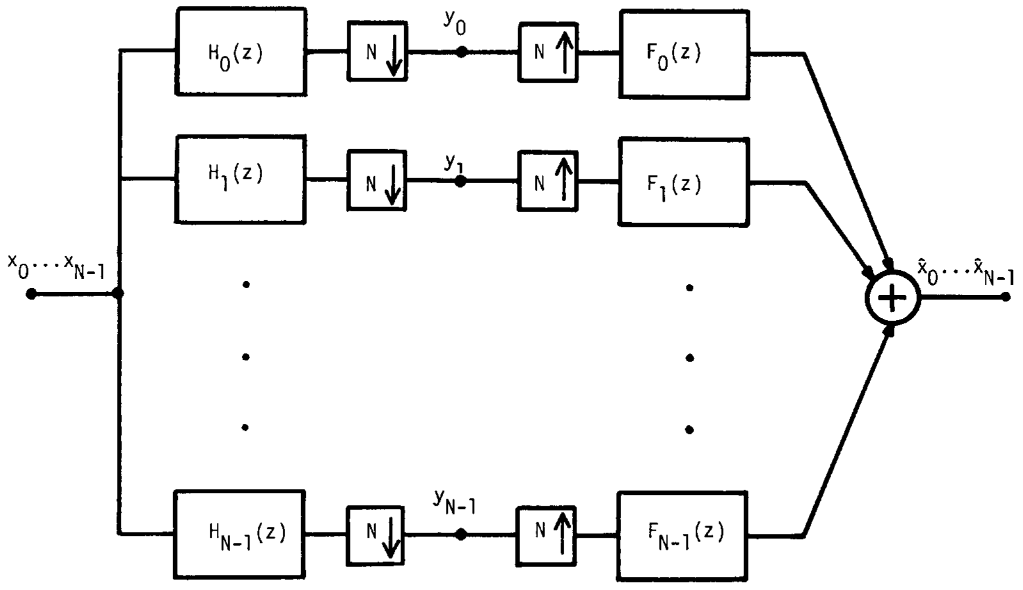

Fig. 2. General decimation/interpolation tree. H_i and F_i are FIR filters of length N.

We turn now to the particular case of binary orthogonal trees. Consider the system of Fig. 3 where the outputs y_0, $y_1 \cdots y_{2N-1}$ are computed every $2N$ samples. The proof that the $2N$ dimensional transform is orthogonal follows the one done for delay trees. The overall transform is given by (11).

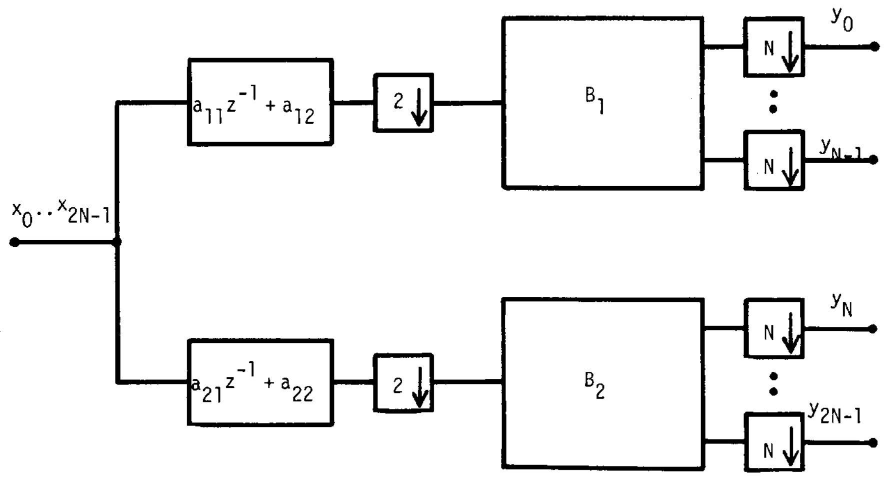

Fig. 3. Decimation tree. B_1 and B_2 are N-dimensional orthogonal transforms.

$$C=\begin{bmatrix} B_1 & \vdots & 0_N \\ \cdots & & \cdots \\ 0_N & \vdots & B_2 \end{bmatrix}\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ 0 & 0 & a_{11} & a_{12} & \cdots & 0 \\ \vdots & & & & & \vdots \\ 0 & 0 & & \cdots & a_{11} & a_{12} \\ a_{21} & a_{22} & 0 & 0 & \cdots & 0 \\ 0 & 0 & a_{21} & a_{22} & \cdots & 0 \\ \vdots & & & & & \vdots \\ 0 & 0 & & \cdots & a_{21} & a_{22} \end{bmatrix} \tag{11}$$

The orthogonality is verified in (12).

$$C\cdot C^{\mathrm{T}}=\begin{bmatrix} B_1 & 0_N \\ 0_N & B_2 \end{bmatrix}\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ 0 & 0 & a_{11} & a_{12} & \cdots & 0 \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ 0 & 0 & & \cdots & a_{11} & a_{12} \\ a_{21} & a_{22} & 0 & 0 & \cdots & 0 \\ 0 & 0 & a_{21} & a_{22} & \cdots & 0 \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ 0 & 0 & \cdots & & a_{21} & a_{22} \end{bmatrix}\cdot\begin{bmatrix} a_{11} & 0 & \cdot & 0 & \cdot & a_{21} & 0 & \cdot & 0 \\ a_{12} & 0 & \cdot & \cdot & \cdot & a_{22} & 0 & \cdot & 0 \\ 0 & a_{11} & & \cdot & \cdot & 0 & a_{21} & & \cdot \\ 0 & a_{12} & \cdot & \cdot & \cdot & 0 & a_{22} & & \cdot \\ \cdot & & & \cdot & \cdot & & & & \cdot \\ \cdot & & & \cdot & \cdot & & & & \cdot \\ \cdot & & & \cdot & \cdot & & & & \cdot \\ \cdot & & & \cdot & \cdot & & & & \cdot \\ \cdot & & & \cdot & \cdot & & & & \cdot \\ \cdot & & a_{11} & 0 & \cdot & & & a_{21} & 0 \\ \cdot & & a_{12} & 0 & \cdot & & & a_{22} & 0 \\ \cdot & & 0 & a_{11} & \cdot & & & 0 & a_{21} \\ 0 & \cdot & 0 & a_{12} & \cdot & 0 & \cdot & 0 & a_{22} \end{bmatrix}$$

$$\cdot\begin{bmatrix} B_1^{\mathrm{T}} & 0_N \\ 0_N & B_2^{\mathrm{T}} \end{bmatrix}=\begin{bmatrix} B_1 & 0_N \\ 0_N & B_2 \end{bmatrix}\cdot\begin{bmatrix} \lambda_1 I_N & 0_N \\ 0_N & \lambda_2 I_N \end{bmatrix}\cdot\begin{bmatrix} B_1^{\mathrm{T}} & 0_N \\ 0_N & B_2^{\mathrm{T}} \end{bmatrix}=\Lambda_C. \tag{12}$$

The initial step is again trivial and the necessity of orthogonality obvious.

Thus, a binary decimation tree performs an orthogonal transform if and only if each subblock is orthogonal.

Above results can be generalized to arbitrary radices, thus for mixed radices as well. This is not done here since only applications of the binary tree will be considered, but the simple condition of orthogonality for branches with a common root is fundamental.

Considering the computational complexity, we restate some well known results. For a radix-K tree working on length N data blocks, the number of operations is given by (13). Here, radix-K means that all nodes (except the terminal ones) are followed by K branches.

$$\begin{aligned} &\text{No. of multiplication} = K * N * \operatorname{Log} N, \\ &\text{No. of additions} \quad = (K-1) * N * \operatorname{Log} N. \end{aligned} \tag{13}$$

Above results where shown for the real case but the complex case follows immediately if one replaces all the transposes by hermitian transposes.

Finally, note that if orthogonal trees generate orthogonal transforms, the reciprocal statement is not true.

3. Walsh–Hadamard Transform computed with tree structures

The tree structures presented above are now used to compute the Walsh–Hadamard transform. When computing the WHT, one is interested by 3 features: number of operations, memory requirement and ordering of the transform output. The number of adds/substracts is always $N \operatorname{Log}_2 N$ for fast WHT's and the memory requirement varies from N to $2N$ for different algorithms.

For the ordering, the definitions as in [5] are used, but note that other definitions are common [6] (namely: natural is called Hadamard, sequency is called Walsh and dyadic is called Paley ordering). For the numbering of the Walsh functions and series, the original definition by Walsh has been used where W_i is a Walsh function with i zero crossings or a Walsh series with i sign changes [7]. Similarly, Y_i is the i-th coefficient of the Hadamard transform in sequency ordering, which equals the scalar product between W_i and the data sequence.

Delay tree for Walsh–Hadamard Transform

The fundamental idea behind tree structured filters is to group common zeros (FIR case) and poles (IIR case) of the various transfer functions in order to reduce the number of computations. The case of the DFT is extensively analysed in [2] where the different zeros of the polynomial $(z^N - 1)$ are grouped in an adequate way [8].

A polynomial definition of the WHT in recursive form as in (14) leads to a filter structure whose output is a running WHT. We call $W(M, i)$ the i-th filter for a transform of dimension 2^M.

$$\begin{aligned} W(M+1, i) &= (z^{-2^M} + 1) * W(M, i), \\ W(M+1, 2^M + i) &= (z^{-2^M} - 1) * W(M, i). \end{aligned} \tag{14}$$

Starting with $W(0, 0) = 1$, we get the Hadamard filters in natural order. The output of the filter $W(M, i)$ is the i-th coefficient of a Hadamard transform in natural order on the 2^M last input samples. Since the filters are divided into two classes having a common factor, one can easily deduce a binary filter tree as shown in Fig. 4.

The computational load for one set of Hadamard coefficients is $N \operatorname{Log}_2 N$. Note that this is an efficient running transformer, since it uses $2N - 2$ operations per input value.

Finally, the impulse response of the system in Fig. 4 is the set of Walsh functions in natural order (with reversed time axis) and thus, the system could also be used as a generator (the same holds for the DFT tree as well).

Decimation tree for the Walsh–Hadamard Transform

A recursive definition of the Walsh series is introduced. Given the set of 2^M Walsh series of length 2^M, $\{W(M, i)\}$, one can deduce the set of 2^{M+1} Walsh series of length 2^{M+1}, $\{W(M+1, i)\}$, by expanding each series into two series of double length as follows:

$$\begin{aligned} W(M+1, 2i) &= [W(M, i), W(M, i)], \\ W(M+1, 2i+1) &= [W(M, i), -W(M, i)]. \end{aligned} \tag{15}$$

Starting with $W(0, 0) := [1]$, we get the Walsh series in dyadic order. It is obvious that:

$$W(M, i) * W(M, j) = 2^M \cdot \delta_{ij} \tag{16}$$

where $*$ is the scalar product and δ_{ij} the Kronecker delta. This orthogonality property together with the fact that each series starts with a 1 defines uniquely the Walsh series.

The recursive definition introduced above leads to a very simple structure for the computation of

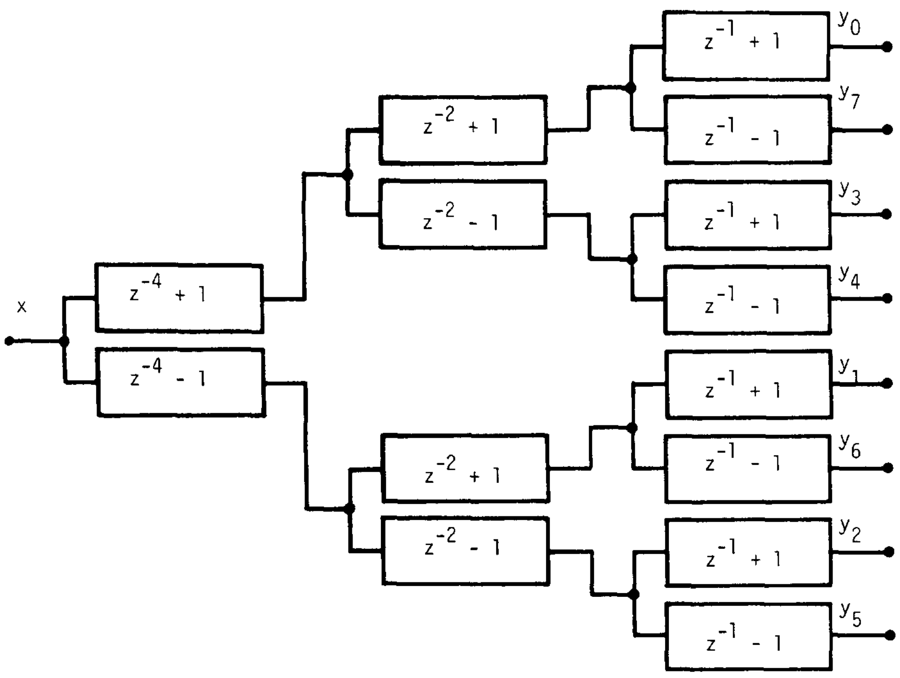

Fig. 4. Binary delay tree for Walsh–Hadamard transform in natural (or Hadamard) order.

the Hadamard transform. The basic building block is a 2-point Hadamard transformer as depicted in Fig. 5.

Starting with this block and building a binary tree of depth M with 2^M ending branches allows the computation of a Hadamard transform of order 2^M on the input data. Because of the subsampling, the result appears every 2^M samples, thus the transform is computed on successive blocks of data. The output is in dyadic order. An 8-point example is shown in Fig. 6.

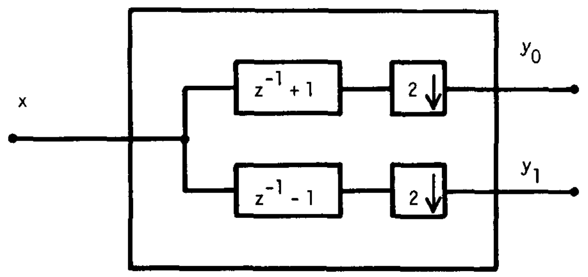

Fig. 5. Basic 2-point Walsh–Hadamard transformer.

Reconstruction is performed with a similar structure, with a change of sign of the differentiator and upsampling by 2. The reconstruction corresponding to Fig. 6 is shown in Fig. 7.

Instead of getting the Hadamard coefficients in dyadic order, one might rather want the sequency ordering. This is obtained when all the blocks on lower branches are reversed, which means that the substractor is above the adder.

Again, one can devise a simple Walsh series generator using upsamplers and interpolators. This is shown in Fig. 8 for a 8-dimensional system.

It is of interest to look at computational and memory requirements. Due to the subsampling, the basic add and subtract operation on 2 successive samples has only to be performed every second sample. At each stage, the clock frequency is reduced by a factor of 2 but at the same time the number of modules is doubled. This results in N adds and subtracts per stage and N input samples. Since there are $\mathrm{Log}_2 N$ stages, the number of

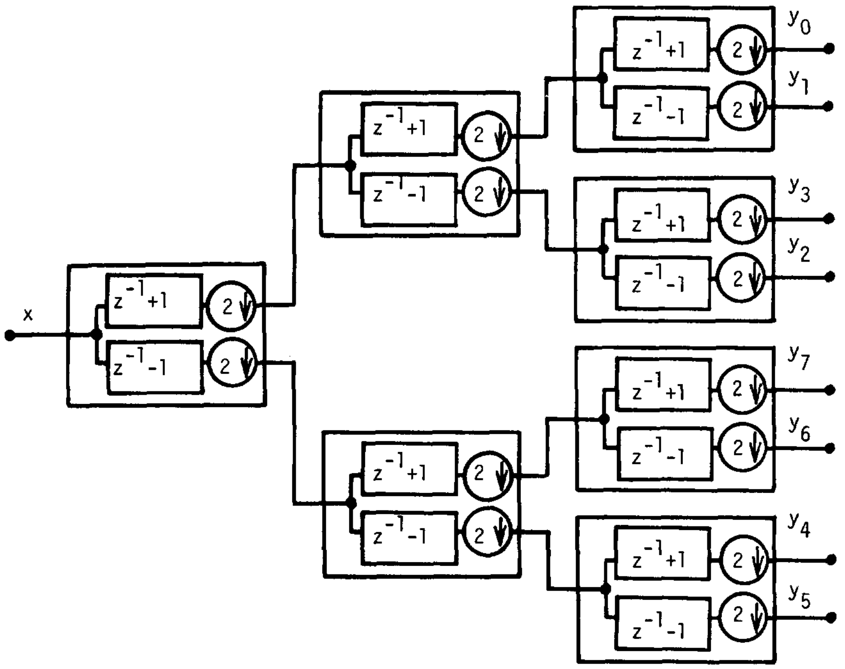

Fig. 6. 8-point Walsh–Hadamard transformer in dyadic (or Paley) order.

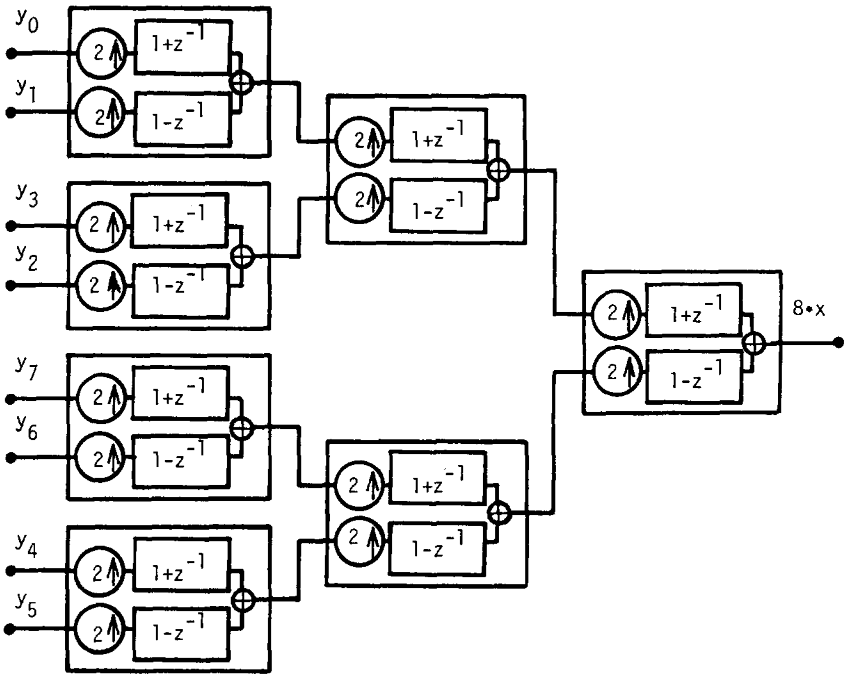

Fig. 7. Reconstruction from the dyadic decomposition.

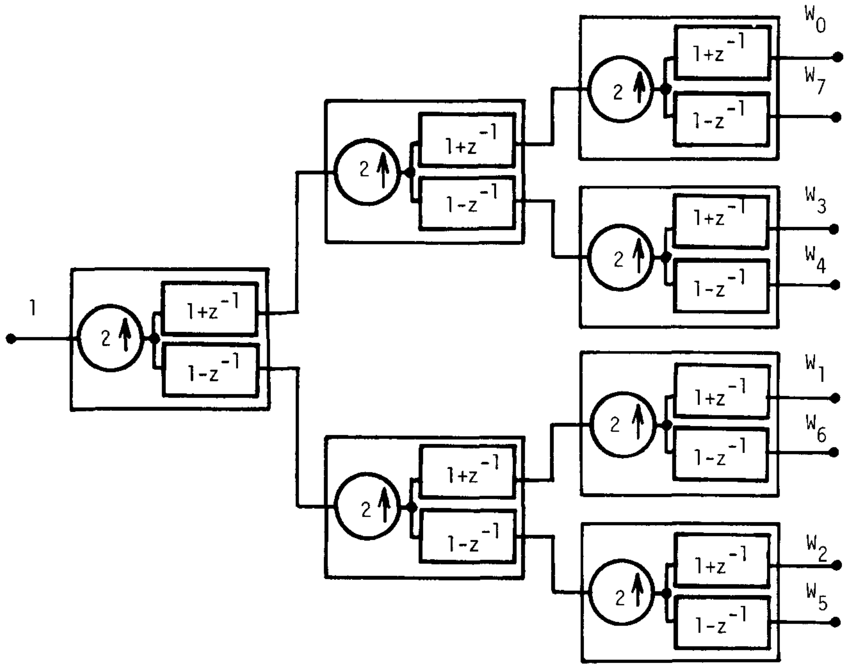

Figure 8. Synthetiser of Walsh series in natural (or Hadamard) order.

operations results in:

$$\text{Number of adds and subtracts} = N \operatorname{Log}_2 N \tag{17}$$

Assuming that each module has a double memory and since there are $N = 1$ modules, we have:

$$\text{Required memory} = 2N - 2. \tag{18}$$

Reordering the data after each stage could allow a memory requirement of $N + 1$ but would substantially slow down the computation.

Note that the subsampling is a fundamental property of trees. Since the transformation is invertible, each stage carries the same amount of information and the sampling frequency can be reduced as the number of branches grows.

4. On the hardware implementation of the Walsh–Hadamard Transform

Some hardware architectures motivated by the tree approach are explored, leading to parallel, multiplexed and pipelined structures. It is assumed that the data is running at a clock frequency f_d and that we compute the WHT on successive blocks at a rate $f_t = f_d/N$.

So far, we have looked at the tree structures from the point of view of computational complexity, memory requirement and transform ordering. Another criterion which has gained in importance with the growing concern about hardware implementation and integration is the structural complexity of the algorithms [9]. New questions are: is the system modular, is the cascading property satisfied and what is the interconnection complexity.

Comparing the decimation tree of Fig. 6 with the equivalent butterfly of Fig. 9 shows the simplicity of the tree approach. Modularity is obvious, since all elements are identical and cascading is therefore trivial (for example, a 64-point transform is obtained if 8 trees as in Fig. 6 are added behind the first tree). Finally, there are no crossings in the interconnection scheme.

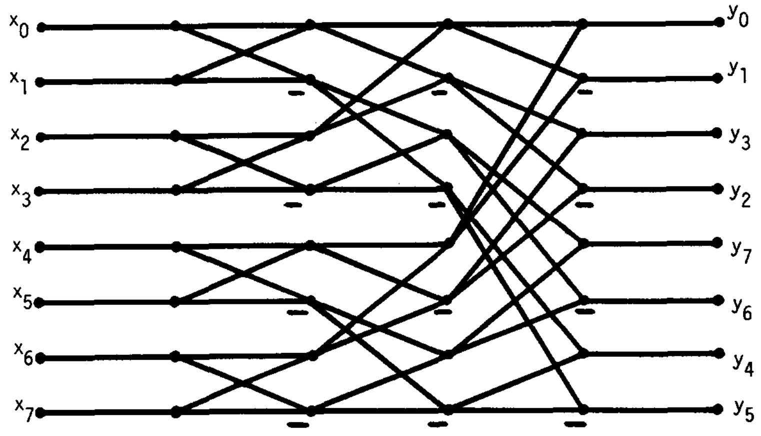

Figure 9. Butterfly configuration for dyadic ordered Walsh–Hadamard transform.

Two properties of the Hadamard decimation tree are now explored under the point of view of data flow. First, it is clear that each module separates the data into two flows which have no interaction further down the tree. Therefore, both flows can be processed separately. Second, the data flow and the processing load at each stage are constants as seen in (19):

$$\begin{aligned}&(\text{sampling frequency}) * (\text{No. of branches}) = \\ &\qquad\qquad\text{constant} \\ &\text{computation power per stage} = \text{ops./sec.} = \\ &\qquad\qquad\text{constant}\end{aligned} \tag{19}$$

Since the complete tree consists of $N-1$ modules as shown in Fig. 6, a straightforward implementation requires $N-1$ separate modules running at different frequencies. Alternatively, a fast module can be multiplexed between the different stages.

In the following, we call a processor a 2-point Hadamard transformer (thus one adder and one subtractor) with 2 input and 2 output memories. The computational power is characterized by f_p (data coming in at a clock frequency f_p can be processed, and the transform appears at a frequency $f_{p/2}$). 3 cases are possible.

Multiplexing case: $f_p \geqslant f_d$

Assume a processor working at $f_p = f_d$. This means that each stage of the tree can be handled with a single processor which is simply multiplexed between the branches. The data from the adder

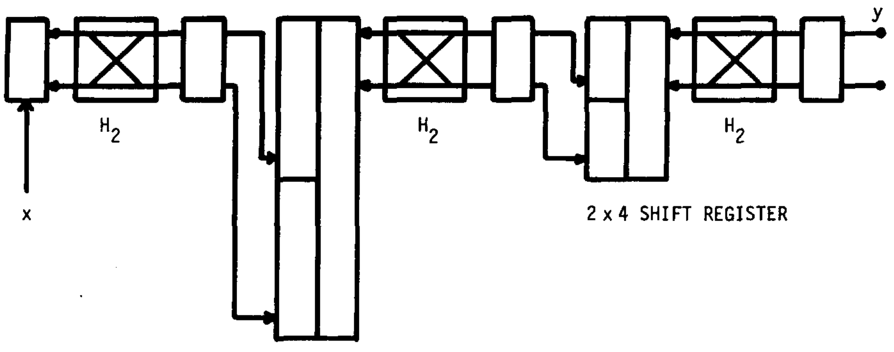

Fig. 10. 8-point dyadic ordered Walsh–Hadamard transformer with a single multiplexed processor per stage.

and subtractor output are stored separately and then processed serially. This can be done with a double shift register or a double length memory with adequate pointers. A 8-dimensional case is shown in Fig. 10 where the output is in Dyadic order. The number of processors is $\mathrm{Log}_2 N$ and the required memory for temporary storage is $4N-8$ which is determinant for the delay of the output transform.

Parallel case: $f_p \leq f_t$

The processing load has to be shared between several processors. Assume $f_p = f_t$. Thus, one processor assumes the load of one module of the last stage, and preceding modules are implemented with several processors. One possible configuration is shown in Fig. 11 where all stages are identical and the output is in natural order. The number of processors is $N/2\,\mathrm{Log}_2 N$ and no buffering memory is required.

Hybrid case: $f_d \geq f_p \geq f_t$

It was seen that the multiplexing case requires a lot of additional memory and that the parallel case leads to numerous additional crossings. If the processor runs at a medium rate, it will be used in parallel at the input and multiplexed at the output of the tree. In Fig. 12, a 64-point transform is computed using 12 4-point transformers (dyadic ordered). The first stage works in parallel on 16-point blocks, the second stage works synchronously and the last stage is multiplexed between 4 branches.

An example

Based on the multiplexing case, a simple module is proposed which can easily be cascaded or linked to itself (see Fig. 13). Each module consists of a processor (2-point Hadamard transformer) and a $2N$ memory. Addresses for all modules are simply generated by 2 counters since write and read addresses are given by (20) where i is the clock:

$$\begin{aligned}
&\text{read adr.:} && 2i \,\mathrm{Mod}\, 2N,\; 2i+1 \,\mathrm{Mod}\, 2N \\
&\text{write adr.:} && N+(i \,\mathrm{Mod}\, N),\; 3/2 * N+(i \,\mathrm{Mod}\, N) \\
& && \text{if } i \,\mathrm{Mod}\, N < N/2 \\
& && i \,\mathrm{Mod}\, N,\; N/2+(i \,\mathrm{Mod}\, N) \\
& && \text{else}
\end{aligned} \tag{20}$$

A 1024-point transform can be performed with 10 cascaded modules or by recirculating the data 10 times through a single one. The output is in natural order. Slowest component will be the RAM for temporary storage, but an access time of 100 ns

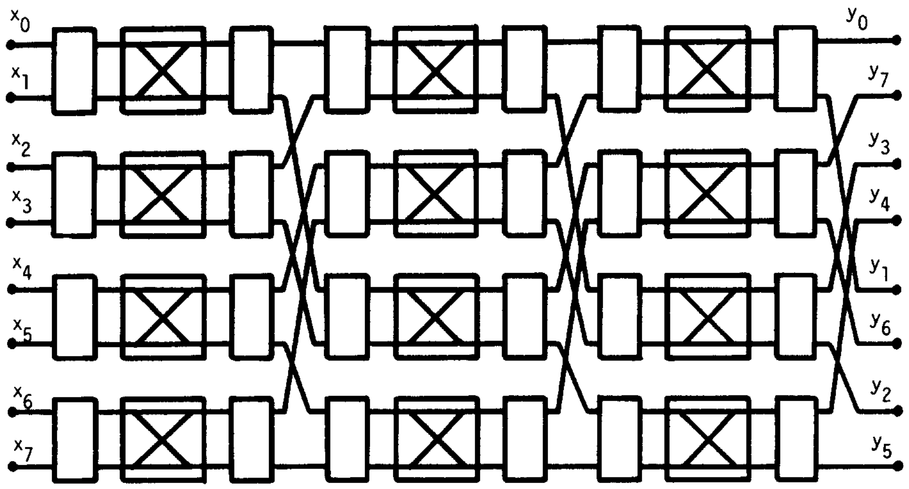

Fig. 11. 8-point natural ordered Walsh–Hadamard transformer with processors used in parallel.

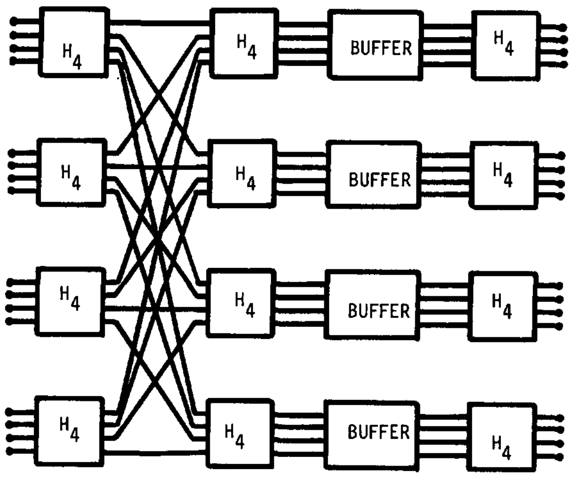

Fig. 12. 64-point Walsh–Hadamard transformer in dyadic order computed with 4-point basic modules. The modules are used in parallel in the first stage and multiplexed in the last stage.

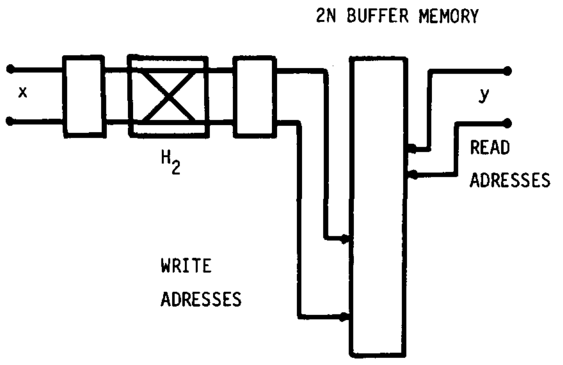

Fig. 13. Basic module for a N-dimensional Walsh–Hadamard transform in natural order.

still leads to a system which can process data at 10 MHz.

5. Conclusion

Three structures defining a class of orthogonal transforms of complexity $N \operatorname{Log}_2 N$ were introduced. It was proved that a necessary and sufficient condition for obtaining an orthogonal transform is the orthogonality of branches with a common root.

Application to the Walsh–Hadamard transform results in simple systems with identical modules and which can easily be cascaded. These structures suggest equivalent hardware taking advantage of the tree features.

The filter tree leads to a running transform with $2N-2$ operations.

This novel approach to the Walsh–Hadamard transform in terms of trees and data flows produces some interesting configurations for fast transformers.

Acknowledgments

The author wishes to thank Prof. H.J. Nussbaumer for making this research possible, C. Petitpierre for stimulating input on hardware structures, A. Schlageter for editing assistance and W. Schmürz for his constant and enthusiastic support.

References

[1] N. Ahmed and K.R. Rao, *Orthogonal Transforms for Digital Signal Processing*, Springer Verlag, Berlin, 1975, Ch. 7, pp. 156–159.

[2] G. Bruun, "z-Transform DFT filters and FFT's", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. 26, No. 1, Feb. 1978, pp. 56–63.

[3] G. Strang, *Linear Algebra and Its Applications*, Academic Press, New York, 1980, Ch. 3, pp. 123–128.

[4] M. Bellanger, *Traitement Numérique du Signal*, Masson, Paris, 1981, Ch. 3, pp. 87–89.

[5] M. Kunt, "In place computation of the Hadamard transform in Cal-Sal order", *Signal Processing*, Vol. 1, No. 3, July 1979, pp. 227–231.

[6] N. Ahmed, H.H. Schreiber, and P.V. Lopresti, "On notation and definition of terms related to a class of complete orthogonal functions", *IEEE Trans. on Electromag. Compat.*, Vol. 15, May 1973, pp. 75–80.

[7] K.G. Beauchamp, *Walsh Functions and Their Applications*, Academic Press, London, 1975, Ch. 2, pp. 17–20.

[8] H.J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*, Springer, Berlin, 1982, Ch. 4, pp. 104–107.

[9] Y.A. Geadah, and M.J.G. Corinthios, "Natural, dyadic, and sequency order algorithms and processors for the Walsh–Hadamard transform", *IEEE Trans. on Computers*, Vol. C-26, No. 5, May 1977, pp. 435–442.